Réduction par symétrie du graphe des classes d'états des Réseaux de Petri Temporels

Pierre-Alain Bourdil[1,2,4], Bernard Berthomieu[1,2],
Silvano Dal Zilio[1,2], and François Vernadat[1,3]

[1] CNRS, LAAS, 7 avenue du Colonel Roche, F-31400 Toulouse, France
[2] Univ de Toulouse, LAAS, F-31400 Toulouse, France
[3] Univ de Toulouse, INSA, LAAS, F-31400 Toulouse, France
[4] Thales Avionics, 105 av du Général Eisenhower, F-31100 Toulouse, France

Abstract. Nous proposons une méthode d'exploitation des symétries pour la vérification par model-checking des systèmes temps réel modélisés par réseaux de Petri temporels. La méthode traite les marquages ainsi que les contraintes temporelles; elle est applicable aux graphes de classes d'états. Nous présentons une implémentation dans TINA ainsi que quelques résultats expérimentaux.

1 Introduction

La méthode de réduction par symétrie exploite les symétries d'un système pour minimiser le nombre d'états à explorer. Cette méthode permet de n'explorer que les classes d'équivalences des états pour la relation de symétrie. Lorsque les applications ont des structures symétriques, ce qui est généralement le cas pour des applications de taille importante, la réduction par symétrie peut contribuer à la réduction de l'explosion combinatoire.

Depuis les premiers travaux exploitant les symétries pour la vérification de programmes ou de réseaux de Petri de haut niveau, les symétries ont été utilisées dans de nombreux contextes. Plusieurs outils supportent les symétries, inférées ou structurelles [16, 8, 21, 20]. La réduction par symétrie des réseaux de Petri peut être combinée avec d'autres techniques de réduction telle que les "stubborn sets" ou les "covering steps".

Dans cet article nous proposons une méthode de réduction par symétrie appliquée à l'abstraction des classes d'états (SCG) pour les réseaux de Petri temporels (Time Petri Nets) [5, 2]. La particularité des modèles temporisés est que leurs espaces d'états sont en général infinis; on construit une représentation finie de ces espaces d'états par une abstraction des contraintes temporelles. Nous manipulons des états composites, ou *classes d'états*, qui capturent à la fois l'information discrète et l'information temporelle. Cette dernière est représentée par des polyèdres. Une technique de réduction par symétrie des classes d'états doit traiter ces deux composantes. Si le traitement des symétries pour la partie discrète (marquages) est bien connu, peu de travaux existent pour la partie temporelle. Des résultats existent toutefois dans le contexte des automates

temporisés pour certains graphes de zones dont les zones obéissent à une propriété dite "diagonale" [14]. Une autre approche, combinant exploration symbolique et symétries, est proposé dans l'outil RED [24]. Étonnamment, alors que l'abstraction des classes d'états est plus ancienne que les graphes de zones, aucune méthode de réduction par symétrie n'a été proposée pour cette construction.

2 Réseaux de Petri Temporels

Un *Réseau de Petri Temporel* (ou TPN) [19], est un réseau de Petri où chaque transition est associée à un intervalle de temps. Cet intervalle contraint les dates possibles de tir de la transition.

Soit $\mathbb{I}$ l'ensemble des intervalles réels non vides à bornes rationnelles non négatives. Un TPN est un tuple $\langle P, T, \mathbf{Pre}, \mathbf{Post}, m_0, I_s \rangle$ où :

- $\langle P, T, \mathbf{Pre}, \mathbf{Post}, m_0 \rangle$ est un réseau de Petri, P est l'ensemble des places, T l'ensemble des transitions, $m_0 : P \to \mathbb{N}$ le marquage initial et $\mathbf{Pre}$, $\mathbf{Post} : T \to P \to \mathbb{N}$ les fonctions préconditions et postconditions.
- $\mathbf{I_s} : \mathbf{T} \to \mathbb{I}$ est une fonction appelée intervalle statique.

2.1 Sémantique d'un TPN

Un marquage est une fonction $m : P \to \mathbb{N}$. Une transition $t \in T$ est *sensibilisée* par m si et seulement si $m \geq \mathbf{Pre}(t)$. Nous noterons $\mathcal{E}(m)$ l'ensemble des transitions sensibilisées par m.

Pour $i \in \mathbb{I}$, $\downarrow i$ désigne sa borne inférieure, et $\uparrow i$ sa borne supérieure (ou ∞ si i n'est pas borné). Pour tout $\theta \in \mathbb{R}_{\geq 0}$ on définit $i \dot{-} \theta = \{x - \theta | x \in i \wedge x \geq \theta\}$.

Un *état* d'un TPN est une paire $s = (m, I)$ où M est un marquage et $I : T \to \mathbb{I}$ est une fonction partielle, appelée fonction intervalle (dynamique) associant un intervalle temporel à chaque transition sensibilisée par m.

Definition 1. *La sémantique d'un TPN $\langle P, T, \mathbf{Pre}, \mathbf{Post}, m_0, I_s \rangle$ est donnée par un système de transitions temporisé $SG = \langle S, s_0, \to \rangle$ où :*

- *S est l'ensemble des états du TPN;*
- *$s_0 = (m_0, I_0)$ est l'état initial, où m_0 est le marquage initial et I_0 la fonction intervalle statique restreinte aux transitions sensibilisées par m_0;*
- *$\to \subseteq S \times (T \cup \mathbb{R}_{\geq 0}) \times S$ est la relation de transition entre états, définie comme suit pour tout $t \in T$ et $\theta \in \mathbb{R}_{\geq 0}$:*
  - *(i) $(m, I) \xrightarrow{t} (m', I')$ si et seulement si:*
    1. *$m \geq \mathbf{Pre}(t)$*
    2. *$m' = m - \mathbf{Pre}(t) + \mathbf{Post}(t)$*
    3. *$0 \in I(t)$*
    4. *$(\forall k)(m' \geq \mathbf{Pre}(k) \Rightarrow$*
       $I'(k) = si\ k \neq t \wedge m - \mathbf{Pre}(t) \geq \mathbf{Pre}(k)\ alors\ I(k)\ sinon\ I_s(k))$

(ii) $(m, I) \xrightarrow{\theta} (m, I')$ *si et seulement si:*
5. $(\forall k)(m \geq \mathbf{Pre}(k) \Rightarrow \theta \leq \uparrow I(k) \wedge I'(k) = I(k) \dot{-} \theta)$

Les transitions discrètes sont étiquetées par un élément de T (cas (i)) tandis que les transitions continues (cas (ii)) sont étiquetées par un réel non négatif. Une transition t peut être tirée depuis l'état (m, I) si t est sensibilisée par m et *immédiatement* franchissable. Dans l'état destination, les transitions k qui sont restées sensibilisées pendant le tir de t (t exclue) conservent leurs intervalles; k est dite *persistante*. Les autres transitions, dites *nouvellement sensibilisées*, sont associées à leurs intervalles statiques.

Une transition continue de θ est possible depuis (m, I) si et seulement si θ n'est pas plus grand que $\uparrow I(k)$ pour tout $k \in \mathcal{E}(m)$. Comme il peut y avoir une infinité de transitions continues, l'espace d'état d'un TPN est généralement infini, même lorsque le réseau de Petri sous-jacent est borné. Dans la section suivante nous décrivons une abstraction finie des ces graphes d'états infinis.

2.2 L'abstraction des classes d'états (Construction SCG)

Un Graphe de Classes d'états (SCG) est une représentation finie du SG qui préserve les marquages et les traces d'un TPN.

Les intervalles I dans les états (m, I) peuvent être vus comme des domaines de tir: le domaine de tir défini par I est l'ensemble des vecteurs réels $(\phi_{t_1}, \ldots, \phi_{t_l})$ où $E = t_1, \ldots, t_l$ sont les transitions sensibilisées par m et, pour tout $i \in E$, nous avons $\phi_i \in I(i)$. Une classe d'état C est un tuple (m, D) où m est un marquage et D un domaine de tir caractérisé par un système d'inéquations linéaires fini. Soit deux classes $C = (m, D)$ et $C' = (m', D')$; deux classes sont égales, noté $C \cong C'$, si et seulement si $m = m'$ et $D \Leftrightarrow D'$ (D et D' ont le même ensemble de solutions).

La construction du graphe des classes [5, 2] consiste à construire inductivement l'ensemble des classes C_σ, pour $\sigma \in T^*$. Intuitivement, la classe C_σ collecte tous les états atteignables par des chemins temporisés tirant les transitions discrètes de σ.

Algorithme 1 (Construction du Graphe des Classes d'états (SCG))
Le SCG est le plus petit ensemble de classes $(C_\sigma)_{\sigma \in T^*}$ *tel que:*

- $C_\epsilon = (m_0, D_0)$ *est la classe initiale où* D_0 *est le domaine défini par l'ensemble d'inégalités* $\{\downarrow I_s(t) \leq \phi_t \leq \uparrow I_s(t) \mid t \in \mathcal{E}(m)\}$.
- *Si* $C_\sigma = (m, D)$ *existe,* t *est sensibilisée par* m*, et le domaine* $D_t = D \wedge \{\phi_t \leq \phi_k \mid t \neq k, k \in \mathcal{E}(m)\}$ *est consistant (auquel cas nous savons que* t *peut être tirée), alors ajouter la classe* $C_{\sigma.t} = (m', D')$ *telle que:*
  (i) m' *est le marquage* $m' = m - \mathbf{Pre}(t) + \mathbf{Post}(t)$*;*
  (ii) D' *est obtenu comme suit depuis* D_t*. Pour chaque transition* $k \in \mathcal{E}(m')$*, ajouter les inéquations* $\phi'_k = \phi_k - \phi_t$ *si* $k \neq t$ *et* k *est persistante (*ϕ'_k *est une nouvelle variable), ou* $\downarrow I_s(k) \leq \phi'_k \leq \uparrow I_s(k)$ *sinon (*k *est nouvellement sensibilisée). Finalement, éliminer du système résultant toutes les variables* ϕ_i.

La construction décrite dans l'Algorithme 1 montre clairement que les domaines de tirs des classes sont des *systèmes de contraintes de différences*, ou plus simplement *systèmes de différences*, c'est à dire des systèmes dans lesquels les inéquations ont la forme $x_i \geq \alpha_i$, $x_i \leq \beta_i$ ou $x_i - x_j \leq \gamma_{i,j}$ et les α_i, β_i et $\gamma_{i,j}$ sont des constantes rationnelles.

Le SCG est une abstraction des espaces d'états des TPN largement utilisée. Elle préserve les marquages et les traces de l'espace d'état, et donc permet le model-checking de LTL. Le SCG est fini si et seulement si le TPN est borné.

Une variante utile du graphe des classes SCG est le *graphe des classes sous inclusion*, noté $SCG_{\subseteq}$ [4]. Il est construit comme le SCG mais toute classe C dont le domaine de tir est inclus dans celui d'une classe C' est identifiée à C'. Cette construction ne préserve que les marquages mais produit un ensemble de classes qui peut être beaucoup plus petit que celui du SCG.

3 Réduction par symétries pour les TPN

Dans cette section, nous adaptons aux TPN les idées proposées par [22] pour les réseaux de Petri.

Definition 2. *Une* symétrie *d'un TPN $\mathcal{N}$ est une permutation π de $P \cup T$ qui préserve le type des noeuds, les préconditions, les postconditions et les intervalles statiques. Plus formellement :*

1. $(\forall x)(x \in P \Leftrightarrow \pi(x) \in P)$
2. $(\forall t, p)(\mathbf{Pre}(t)(p) = \mathbf{Pre}(\pi(t))(\pi(p)))$
3. $(\forall t, p)(\mathbf{Post}(t)(p) = \mathbf{Post}(\pi(t))(\pi(p)))$
4. $(\forall t)(I_s(t) = I_s(\pi(t)))$

L'ensemble $S_{\mathcal{N}}$ des symétries de $\mathcal{N}$ forme un sous-groupe du groupe symétrique $Sym(P \cup T)$. De plus $S_{\mathcal{N}}$ induit des symétries sur l'espace d'état de $\mathcal{N}$.

Soit (m, I) un état d'un TPN (m est un marquage et I une fonction intervalle). Si $\pi \in S_{\mathcal{N}}$ est une symétrie d'un TPN, alors :

- L'action $\pi(m)$ de π sur m est définie par $(\forall p \in P)(\pi(m) = m(\pi^{-1}(p)))$
- L'action $\pi(I)$ de π sur I est définie par $(\forall t)((\pi(I))(t) = I(\pi^{-1}(t)))$
- L'action $\pi(m, I)$ de π sur un état (m, I) est définie par $\pi(m, I) = (\pi(m), \pi(I))$.

Le lemme suivant explicite les symétries induites sur l'espace d'état de $\mathcal{N}$. Sa preuve est immédiate en utilisant la définition des états.

Lemme 1 *Soit π un symétrie d'un TPN $\mathcal{N}$. Nous avons pour tout t, θ, m, m', I, I':*

1. $(m, I) \xrightarrow{t} (m', I) \Leftrightarrow \pi(m, I) \xrightarrow{\pi(t)} \pi(m', I')$
2. $(m, I) \xrightarrow{\theta} (m', I') \Leftrightarrow \pi(m, I) \xrightarrow{\theta} \pi(m', I')$

Soit π une symétrie d'un TPN, nous définissons $\pi(D)$ comme l'ensemble des solutions de D après permutation des variables par π. L'action $\pi(m, D)$ de π sur la classe (m, D) est définie par $\pi(m, D) = (\pi(m), \pi(D))$ et $(m, D) \approx (m', D')$ si et seulement si $(\exists \pi \in S_{\mathcal{N}})(\pi(m, D) = (m', D'))$.

Soit $SCG_{\approx}$ le graphe des classes tel que défini par l'algorithme 1, en considérant les classes modulo $\approx$. Le théorème suivant étend le lemme 1 aux SCG.

Théorème 1 *Soit π une symétrie du TPN $\mathcal{N}$. Alors $(\forall t, m, m', D, D')$:*

1. $(m, D) \xrightarrow{t} (m', D) \Leftrightarrow \pi(m, D) \xrightarrow{\pi(t)} \pi(m', D')$
2. *si m_0 est symétrique alors $c \in SCG \Leftrightarrow (\exists c^* \in SCG_{\approx})(c \approx c^*)$*

Preuve. Omise.

Deux problèmes se posent lors de l'exploitation des symétries : l'identification des symétries du réseau et le test d'équivalence des classes du SCG pour $\approx$.

3.1 Déclarations des symétries

Notre approche d'identification des symétries repose sur la déclaration statique des symétries dans le modèle. L'annotation des modèles pour en déduire ses symétries était déjà utilisée dans Murφ [16] qui introduisait pour cela un type de données spécifique, les "scalarset", aussi utilisés par UPPAAL [14]. Pareillement, les symétries des réseaux de Petri de haut niveau sont déduites de la syntaxe de leurs inscriptions [9].

Dans notre implémentation nous décrivons les réseaux hiérarchiquement, comme des compositions de sous-réseaux. Des opérateurs de composition ad-hoc définissent à la fois l'architecture du réseau et ses symétries.

Les opérateurs de composition symétrique, *Pool* et *Ring*, sont des spécialisations de l'opérateur de composition des TPN [6]. Chacun des opérateurs a deux paramètres, une arité n et un TPN C. $Pool(n, C)$ décrit un produit libre de n copies distinctes du TPN C. Ses symétries sont celles dérivées des transpositions des copies. $Ring(n, C)$ décrit un produit synchronisé de n copies distinctes de C, après un renommage conventionnel assurant la synchronisation d'un composant avec ses voisins. Les symétries sont dérivées des permutations cycliques de l'anneau des copies.

Nous donnons quelques exemples de réseau et de leurs symétries, définis à l'aide de ces opérateurs symétriques :

- $(Ring(8, P) \mid Q \mid Pool(3, R))$ décrit le TPN obtenu en synchronisant 8 instances de P en anneau, avec Q et un pool de 3 instances de R.
- $Ring(6, P \mid Pool(4, Q))$ décrit un anneau de 6 instances du composant $P \mid Pool(4, Q)$. Ce dernier synchronisant P avec un pool de 4 copies de Q.
- $T \mid Pool(3, S1 \mid Pool(3, C1)) \mid (S2 \mid Pool(2, C2))$ décrit une architecture 3-tiers proposée en exemple dans [11], Figure 1. C'est une architecture hiérarchique dont les feuilles sont les clients, les 3 instances de $S1$ ainsi que $S2$ sont des serveurs et la racine T est un serveur de base de données.

Ces descriptions définissent à la fois un TPN et ses symétries structurelles. Ils décrivent des groupes cycliques, des groupes symétriques et des produits disjoints ou couronnes de ces groupes. Ces constructions sont décrites dans [10, 11].

3.2 Exploitation des symétries

Le problème fondamental pour la réduction des espaces d'états est de déterminer si deux états s_1, s_2 sont équivalents, noté $s_1 \approx s_2$. Ce problème est identifié dans [13] comme le *problème de l'orbite* .

Trois méthodes sont généralement proposées pour déterminer si $s_1 \approx s_2$: énumérer les orbites et tester l'appartenance, vérifier constructivement $\approx$ et calculer une forme canonique pour les états. La première méthode n'est viable que lorsque les orbites sont petites (eg. pour les groupes cycliques). La deuxième suppose une représentation particulière des groupes (voir par exemple [17]). La plupart des implémentations, la notre comprise, exploitent la dernière solution : le calcul de formes canoniques.

Soit σ une fonction qui associe à une classe $C = (m, D)$ son représentant. σ est consistante si $\sigma(C) = \sigma(C')$ implique $C \approx C'$. Si de plus $C \approx C'$ implique $\sigma(C) = \sigma(C')$ alors σ calcule une forme canonique. Étant donné un ordre total sur les classes nous définissons σ comme la fonction qui calcule la plus petite classe de l'orbite relativement à cet ordre. Puisque les orbites sont finies il est toujours possible de calculer ce plus petit élément.

Néanmoins, 1) la complexité du calcul d'une forme canonique peut être très élevée et 2) il reste à définir un ordre total sur les classes. Notre expérimentation implémente des solutions efficaces pour 1), inspirées de [10, 11]. La principale difficulté réside dans la définition d'un ordre total pour les classes d'états.

Nous utilisons l'ordre lexicographique sur les marquages des places pour la partie discrète des classes. La partie temporelle est encodée par un système de différences. Un ordre total peut être défini sur de tels systèmes en général mais sa complexité est trop élevée pour une implémentation efficace : le problème se réduit en général à celui de l'isomorphisme de graphes [15]. Nous proposons dans la section suivante un ordre total de complexité quadratique sur les domaines des classes d'états équivalentes.

4 Un ordre total sur les classes d'états équivalentes

Nous définissons un ordre total, noté $\preceq_D$ (ou simplement $\preceq$ si le contexte est non ambigu), entre les transitions sensibilisées dans une classe $C = (m, D)$ et équivalentes par symétrie ($\approx$). Cette relation d'ordre sur les transitions nous permet ensuite de définir un ordre total sur les classes équivalentes.

Intuitivement, $t \preceq_D t'$ si t' a été sensibilisée après t dans l'exécution qui mène à C. Le domaine de tir D de C ne contient pas explicitement l'information nécessaire pour déterminer leurs dernières dates de sensibilisation, mais nous exhibons un invariant sur les domaines de tirs qui permet de reconstruire cette information.

Une idée similaire est utilisée pour les automates temporisés dans [14]. Dans une zone, une horloge h est inférieure à h' si le dernier reset de h' a lieu après le dernier reset de h. Cependant, la construction du graphe des zones pour les automates temporisés diffèrant largement de celle des classes du SCG, le traitement technique est lui aussi très différent.

4.1 Forme normale des domaines de tirs

Pour simplifier l'écriture, les variables des domaines de tirs (notées ϕ_t en section 2) seront simplement notées ici comme les transitions auxquelles elles sont associées. Aussi, pour plus de lisibilité, nous noterons dans cette section $\downarrow I_s(t)$ par α_t^s (la date statique de tir au plus tôt de t) et $\uparrow I_s(t)$ par β_t^s (date statique de tir au plus tard de t).

Les domaines de tirs calculés dans la construction du SCG (Algorithme 1) peuvent toujours être réécrits dans une forme réduite de la forme suivante, pour toutes t et t' sensibilisées par le marquage de la classe:

$$\begin{array}{l} \alpha_t \leq t \leq \beta_t \\ t - t' \leq \gamma_{t,t'} \qquad (t \neq t') \end{array}$$

En déterminant les plus grand α_t et les plus petits β_t et $\gamma_{t,t'}$ qui préservent l'ensemble des solutions de D, nous obtenons la forme normale, ou canonique de D. Les formes normales des domaines de tir peuvent être calculées incrémentalement avec une complexité de $\mathcal{O}(n^2)$ en temps (voir par exemple [7, 23]).

Lemme 2 (Calcul incrémental des domaines de tir) *Soit $C = (m, D)$ une classe, et D en forme normale. Pour toute transition $t \in \mathcal{E}(m)$ il existe une classe unique $C' = (m', D')$ obtenue à partir de C en tirant t telle que D' est aussi en forme normale, et pour toutes les transitions distinctes $i, j \in \mathcal{E}(m')$:*

$$\begin{array}{rlrl} \beta'_i = \beta_i^s & \textit{si } i \textit{ nouv. sensibilisée,} & \beta'_i = \gamma_{i,t} & \textit{sinon} \\ \alpha'_i = \alpha_i^s & \textit{si } i \textit{ nouv. sensibilisée,} & \alpha'_i = \max(0, -\min_{k \in \mathcal{E}(m)}(\gamma_{k,i})) & \textit{sinon} \\ \gamma'_{i,j} = \beta'_i - \alpha'_j & \textit{si } i \textit{ ou } j \textit{ nv. sensibilisée,} & \gamma'_{i,j} = \min(\gamma_{i,j}, \beta'_i - \alpha'_j) & \textit{sinon} \end{array}$$

4.2 Un ordre total entre les transitions équivalentes

Nous prouvons un invariant sur les domaines de tir, en forme normale, obtenu dans la construction du SCG. Il explicite une relation entre les transitions ayant le même intervalle statique (et donc mêmes α^s et β^s), ce qui est le cas pour les transitions équivalentes.

Deux transitions i et j sont équivalentes dans D, noté $i =_D j$, si et seulement si, la transposition de i et j dans le domaine D est un domaine D' ayant le même ensemble de solutions.

Lemme 3 *Si D est en forme normale alors $i =_D j$ si et seulement si $\alpha_i = \alpha_j$, $\beta_i = \beta_j$, $\gamma_{i,j} = \gamma_{j,i}$ et pour toute transition k différente de i, j, $\gamma_{i,k} = \gamma_{j,k} \ \wedge \ \gamma_{k,i} = \gamma_{k,j}$.*

Nous définissons maintenant un ordre $i \preceq_D j$ (i est avant j), par:

$$i \preceq_D j \quad =_{\text{def}} \quad \gamma_{i,j} \leq \gamma_{j,i} \ \wedge \ (\forall k \neq i,j)(\gamma_{i,k} \leq \gamma_{j,k}) \qquad (\preceq\text{-DEF})$$

Lemme 4 *Soit (m, D) une classe du SCG et i, j deux transitions ayant le même intervalle statique ($I_s(i) = I_s(j)$). Alors $i \preceq_D j$ implique $\alpha_i \leq \alpha_j$, $\beta_i \leq \beta_j$ et pour toute transition k différente de i, j, $\gamma_{k,j} \leq \gamma_{k,i}$.*

Preuve. Par induction sur la séquence de tir menant à (m, D) . □

Un corollaire simple du lemme 4 est que la relation $\preceq_D$ est antisymétrique pour les paires de transitions équivalentes: Si $i \preceq_D j$ et $j \preceq_D i$ alors $i =_D j$. Clairement, $\preceq_D$ est aussi reflexif et transitif.

Lemme 5 *Soit (m, D) une classe du SCG et i, j deux transitions sensibilisées par m ayant le même intervalle statique ($I_s(i) = I_s(j)$). Nous avons alors $i \preceq_D j$ ou $j \preceq_D i$*

Preuve. Par analyse des trois cas à considérer. Soit i et j ont été sensibilisée simultanément dans D (elles ont le même âge); soit une des deux seulement est nouvellement sensibilisée; soit les deux sont persistentes. □

$\preceq_D$ ordonne donc totalement les transitions de même intervalle statique dans un domaine de tir, et donc les transitions équivalentes par $\approx$. $\preceq_D$ est aisément étendu aux transitions non sensibilisées en choisissant $i \preceq_D j$ vraie pour toute i non sensibilisée (et $j \approx i$).

5 Calcul des formes canoniques

La forme canonique d'un état est choisie comme l'état de son orbite le plus petit lexicographiquement [10]. Pour vérifier si deux états sont équivalents nous testons l'égalité de leurs formes canoniques. Cette section précise le calcul en temps polynomial de la forme canonique des classes d'états pour les symétries retenues (voir Section 3).

Groupes symétriques Les composants concernés sont des sous-réseaux du TPN, disjoints et isomorphes par construction.

Soit une classe $C = (m, D)$, nous définissons la fonction e qui à chaque transition $t \in T$ associe soit $\perp$ si t n'est pas sensibilisée dans C, soit la variable ϕ_t de D sinon (ϕ_t est la variable du domaine de tir associée à la transition t).

En projetant le marquage m et e sur les composants, on obtient un tuple de paires $((m^1, e^1), ..., (m^n, e^n))$ où m^i est le marquage m restreint aux places du composant i et e^i est la restriction de e aux transitions du composant i.

En choisissant un ordre total $\leq_P$ (resp. $\leq_T$) sur les places (resp. transitions), nous pouvons représenter les marquages m^i et les variables e^i par des vecteurs. L'ordre $\leq_P$ doit être tel que les éléments de même indice dans deux marquages

(vecteurs ici) de composants correspondent à des éléments équivalents par $\approx$. Nous avons une condition similaire pour $\leq_T$ et les variables e^i. Ces conditions sont des conditions nécessaires pour la consistance des comparaisons des états des composants. Dans notre implémentation, elles découlent du nommage des places et transitions des composants.

La fonction de comparaison $\leq_c$, qui compare les états des composants, est ensuite définie comme suit :

$$(m^i, e^i) \leq_c (m^j, e^j) =_{\text{def}}$$
$$(m^i \leq_m m^j \wedge \neg(m^j \leq_m m^i)) \vee (m^i \leq_m m^j \wedge m^j \leq_m m^i \wedge e^i \leq_e e^j)$$

où

$\leq_m$ compare lexicographiquement deux vecteurs de marquages, en utilisant la comparaison d'entiers pour leurs éléments ;
$\leq_e$ compare lexicographiquement deux vecteurs de variables, en utilisant l'ordre $\preceq_D$ pour comparer leurs éléments (défini en Section 4). $\preceq_D$ est relatif au domaine D et, par convention, $\perp$ est toujours plus petit par $\preceq_D$.

$\leq$ sur les entiers et $\preceq_D$ sur les variables des domaines sont des ordres totaux. En conséquence, $\leq_m$, $\leq_e$, et $\leq_c$ le sont aussi.

Pour obtenir la forme canonique d'une classe (m, D) nous ordonnons les états des composants par $\leq_c$, ce qui nous donne une permutation σ des composants. Nous déduisons de σ la permutation π du réseau que nous appliquons ensuite sur la classe (m, D) pour obtenir sa forme canonique.

Autres groupes: Pour les groupes cycliques, nous associons à chaque composant son rang dans la liste des composants ordonnés par $\leq_C$; les composants ayant le même état ayant le même rang. Nous cherchons ensuite la permutation cyclique des composants σ qui minimise lexicographiquement la liste des rangs. Nous déduisons de σ une permutation π du réseau qui, appliquée à la classe (m, D), nous donne sa forme canonique. Le traitement des produits disjoints et couronnes est similaire à celui décrit dans [12]

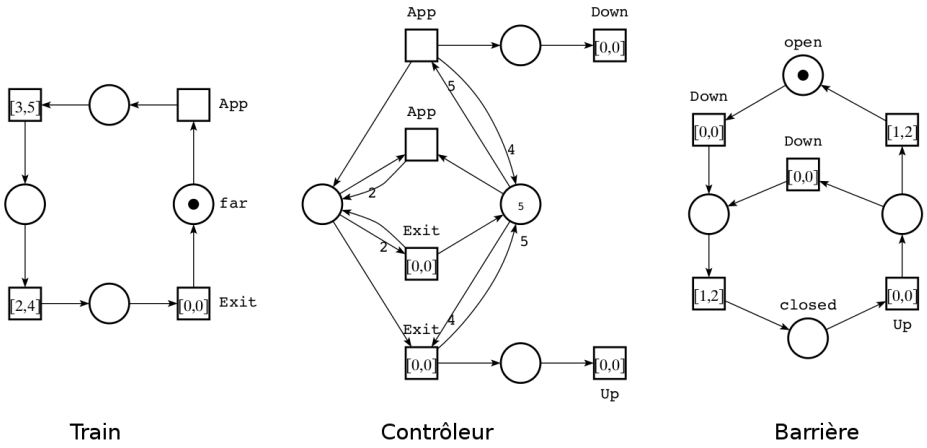
6 Expérimentations

Nous avons implémenté la méthode de réduction par symétrie décrite dans une extension de *TINA* [3], une boite à outil d'analyse des TPN et de certaines extensions des TPN[5]. Nous montrons ci-après quelques résultats expérimentaux obtenus sur l'exemple du passage à niveau tiré de [1].

[5] TINA est disponible à l'adresse http://www.laas.fr/tina. La version supportant les symétries est disponible pour expérimentation à l'adresse http://www.laas.fr/tina/symmetries

6.1 Exemple de symétrie simple, le passage à niveau :

Cet exemple modélise un passage a niveau : un nombre de voies traversent une route, protégées par une barrière; lorsque les trains s'approchent ou s'éloignent de la route, ils déclenchent un signal qui est émis vers un contrôleur. Celui-ci lève ou abaisse la barrière. Nous cherchons à vérifier une propriété de sureté : la barrière doit être fermée dès lors qu'un train traverse la route. Cette propriété est assurée par des contraintes temporelles sur certains événements. Le réseau et les symétries sur les voies sont spécifiés par un script dans une extension du format `tpn` de *TINA*, donné en figure 2 . Le résultat est illustré dans la figure 1.



voies	*SCG*	*SCG*/≈	$SCG_{\subseteq}$	$SCG_{\subseteq}/\approx$
3	3101	578	172	41
4	134501	6453	1175	76
5	8557621	84510	10972	143
6	697913229	1183782	128115	274
7	7.278×10^{10}	18143796	1772722	533
8	9.262×10^{12}	297205635		1048
10				4126
12				16420
14				65578
16				262192
18				1048630

Fig. 1. Passage à niveau temporisé

La table Figure 1 donne les résultats de notre expérimentation pour trois constructions. Les valeurs grisées ont été interpolées en sommant les tailles des orbites du graphe réduit par symétrie. La colonne *SCG* présente les tailles de

l'espace d'état non réduit. La colonne $SCG/\approx$ présente les tailles des espaces d'états réduits. Nous obtenons les gains attendus sur le nombre de classes.

L'abstraction par inclusion de domaines ($SCG_{\subseteq}$) mentionnée en section 2.2, ne préservant que les marquages (mais pas les traces), bénéficie aussi de la réduction par symétrie, comme le montre la colonne $SCG_{\subseteq}/\approx$.

La réduction par symétrie apporte aussi un gain en temps de calcul qui augmente avec la taille de l'espace d'état. En résumé, nos expérimentations confirment que la réduction par symétrie des classes d'états apporte les mêmes bénéfices que lorsque appliquée aux seuls états discrets, elle permet de construire les espaces d'états plus rapidement et avec moins de ressources mémoire.

```
# Initial state is symmetric
# Push a train component on stack
load components/train.ndr
# Push two more copies
dup 2
# Build a pool of 3 instances of train
merge pool 3
# Push the controler
load components/controler3.ndr
# Push the barrier
load components/barrier.ndr
# Synchronize the pool of trains, the controler and the barrier.
sync 3
```

Fig. 2. Construction par composition du Passage à niveau temporisé (3 trains)

7 Travaux similaires et Conclusion

Dans le contexte des réseaux de Petri, la méthode de réduction par symétrie a été principalement appliquée à la classe des réseaux colorés. Il faut aussi noter les travaux de [18] sur les symétries de données et l'utilisation de représentations particulières des groupes pour la résolution du problème de l'orbite. Nos travaux portent sur les modèles temps réels. Les seuls développements sur les symétries pour les modèles temps réel dont nous avons connaissances ont été réalisés pour UPPAAL [14] et RED [24]. Ce dernier exploite les symétries dans une exploration symbolique et repose sur des sur-approximations de l'espace d'état; il utilise donc des méthodes différentes de celle que nous proposons. Le traitement des symétries dans UPPAAL est, dans l'esprit, le plus proche du notre.

Nous avons développé et mis en oeuvre une méthode de réduction par symétrie pour la construction du graphe des classes d'états pour les réseaux de Petri temporels. A notre connaissance aucun résultat n'existe pour cette construction. Le traitement technique proposé en section 4 diffère significativement

de [14]. Cette différence n'est pas surprenante puisque les abstractions pour les automates temporisés sont basées sur des zones d'horloges – c'est à dire sur une abstraction des événements passés – alors que les classes d'états capturent les intervalles de temps futurs auxquels une transition peut être tirée. En pratique les domaines de tirs sont plus intéressants pour les TPN car produisant généralement des graphes de classes plus petits.

Actuellement, nous étudions plusieurs améliorations possibles de notre méthode. Notre approche pragmatique de définition compositionnelle des symétries (section 3) est suffisante dans la plupart des cas mais elle interdit les interactions entre composants d'un pool, ce qui peut être limitant. De plus, nous souhaiterions intégrer d'autres groupes tels que les groupes dihédraux ou le groupe des symétries du cube.

References

1. B. Berthomieu and F. Vernadat. State class constructions for branching analysis of Time Petri Nets. In *TACAS 2003*, page 442. Springer LNCS 2619, 2003.
2. B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using time petri nets. *IEEE Trans. on Software Engineering*, 17(3):259–273, 1991.
3. B. Berthomieu, P.-O. Ribet, and F. Vernadat. The tool TINA – construction of abstract state spaces for Petri nets and time Petri nets. *International Journal of Production Research*, 42(14):2741–2756, 15 July 2004.
4. B. Berthomieu and F. Vernadat. *State Space Abstractions for Time Petri Nets.* Handbook of Real-Time and Embedded Systems, Ed. Insup Lee, Joseph Y-T. Leung and Sang Son, CRC Press, Boca Raton, FL., U.S.A., 2007.
5. Bernard Berthomieu and Miguel Menasche. An enumerative approach for analyzing time petri nets. In *Proc. IFIP 1983*, pages 41–46. Elsevier Science Publishers, 1983.
6. Bernard Berthomieu, Florent Peres, and François Vernadat. Bridging the gap between timed automata and bounded time petri nets. In *Formal Modeling and Analysis of Timed Systems*, pages 82–97. Springer LNCS 4202, 2006.
7. Hanifa Boucheneb and John Mullins. Analyse des réseaux temporels: calcul des classes en o(n[2]) et des temps de chemin en o(mxn). *TSI. Technique et science informatiques*, 22(4):435–459, 2003.
8. Giovanni Chiola. Manual and automatic exploitation of symmetries in SPN models. In *Application and Theory of Petri Nets 1998*, pages 28–43. Springer, 1998.
9. Giovanni Chiola, Claude Dutheillet, Giuliana Franceschinis, and Serge Haddad. On well-formed coloured nets and their symbolic reachability graph. In *High-level Petri Nets*, pages 373–396. Springer, 1991.
10. E. Clarke, E. Emerson, S. Jha, and A. Sistla. Symmetry reductions in model checking. In Alan Hu and Moshe Vardi, editors, *Computer Aided Verification*, pages 147–158. Springer LNCS 1427, 1998.
11. Alastair Donaldson and Alice Miller. Exact and approximate strategies for symmetry reduction in model checking. In J. Misra, T. Nipkow, and E. Sekerinski, editors, *FM 2006: Formal Methods*, pages 541–556. Springer LNCS 4085, 2006.
12. Alastair Donaldson and Alice Miller. On the constructive orbit problem. *Annals of Mathematics and Artificial Intelligence*, 57:1–35, 2009.
13. E. Allen Emerson and A. Prasad Sistla. Symmetry and model checking. *Formal Methods in System Design*, 9:105–131, 1996. 10.1007/BF00625970.

14. M. Hendriks, G. Behrmann, K. G. Larsen, P. Niebert, and F. W. Vaandrager. Adding symmetry reduction to Uppaal. In *Formal Modeling and Analysis of Timed Systems*, pages 46–59. Springer LNCS 2791, 2004.
15. Martijn Hendriks. Enhancing uppaal by exploiting symmetry. Technical report, Nijmegen Institute for Computing and Information Sciences, 2002.
16. C Norris Ip and David L Dill. Verifying systems with replicated components in murϕ. In *Computer aided verification*, pages 147–158. Springer, 1996.
17. T. A. Junttila. New canonical representative marking algorithms for place/transition-nets. In Jordi Cortadella and Wolfgang Reisig, editors, *Applications and Theory of Petri Nets 2004*. Springer LNCS 3099, 2004.
18. T.A. Junttila. New orbit algorithms for data symmetries. In *Application of Concurrency to System Design, 2004. ACSD 2004. Proceedings. Fourth International Conference on*, pages 175 – 184, june 2004.
19. Philip M Merlin and David J Farber. Recoverability of communication protocols–implications of a theoretical study. *Communications, IEEE Transactions on*, 24(9):1036–1043, 1976.
20. Karsten Schmidt. LOLA – A low level analyser. In *Application and Theory of Petri Nets 2000*, pages 465–474. Springer, 2000.
21. A. Prasad Sistla, Viktor Gyuris, and E. Allen Emerson. Smc: a symmetry-based model checker for verification of safety and liveness properties. *ACM Trans. Softw. Eng. Methodol.*, 9(2):133–166, April 2000.
22. Peter H Starke. Reachability analysis of petri nets using symmetries. *Systems Analysis Modelling Simulation*, 8(4-5):293–303, 1991.
23. Enrico Vicario. Static analysis and dynamic steering of time-dependent systems. *IEEE Transactions on Software Engineering*, 27(8):728–748, 2001.
24. Farn Wang and Karsten Schmidt. Symmetric symbolic safety-analysis of concurrent software with pointer data structures. In *Formal techniques for networked and distributed sytems*, pages 50–64. Springer LNCS 2529, 2002.